Uni-FACEF CENTRO UNIVERSITÁRIO MUNICIPAL DE FRANCA

Geovana Cardoso Silva - 25332
Vinicius de Luca Prado - 25793

ALGORITMOS DE BUSCA E ORDENAÇÃO

ESTRUTURA DE DADOS II

FRANCA
2025

Geovana Cardoso Silva - 25332
Vinicius de Luca Prado - 25793

ALGORITMOS DE BUSCA E ORDENAÇÃO

ESTRUTURA DE DADOS II

Relatório de Estrutura de Dados II, apresentado Prof. Alexandre do Uni-FACEF Centro Universitário Municipal de Franca.

Prof. Responsável: **Prof. Alexandre**

FRANCA
2025

SUMÁRIO

1 INTRODUÇÃO

Este trabalho visa documentar o desenvolvimento de uma aplicação em JavaScript para o cadastro e manipulação de dados de alunos de uma faculdade. A aplicação, desenvolvida com a biblioteca React, utiliza uma estrutura de dados dinâmica e heterogênea baseada em Arrays de Objetos para armazenar as informações dos alunos (Nome, RA, Idade, Sexo, Média e Resultado). Além disso, o software é capaz de gerar relatórios ordenados e filtrados utilizando o algoritmo de ordenação Selection-Sort, demonstrando a aplicação prática de conceitos de estruturas de dados e algoritmos. O objetivo é apresentar a solução técnica, justificando as escolhas de implementação e evidenciando a sua funcionalidade.

2 ESTUDOS TEÓRICOS

3 ESTRUTURA DE DADOS UTILIZADA: ARRAY DE OBJETOS

A estrutura de dados central desta aplicação é o **Array de Objetos**. Trata-se de uma estrutura dinâmica, pois o seu tamanho pode ser ajustado em tempo de execução, e heterogênea, pois cada elemento do array (um objeto) pode conter dados de diferentes tipos (string, número, etc.).

Cada aluno é representado por um objeto com as seguintes propriedades:

- `nome`: string
- `ra`: string
- `idade`: number
- `sexo`: string
- `media`: number
- `resultado`: string

3.1 Vantagens

A escolha desta estrutura foi motivada por diversas vantagens:

- **Flexibilidade:** Permite agrupar diferentes tipos de dados de um único aluno em uma unidade coesa (o objeto), tornando o código mais organizado e legível.
- **Acesso por Índice:** É possível acessar qualquer aluno de forma rápida e eficiente usando seu índice no array, o que facilita a manipulação dos dados.
- **Natureza Dinâmica:** O array pode ser expandido para acomodar novos alunos, sem a necessidade de pré-alocar um tamanho fixo, o que o torna ideal para um sistema de cadastro.

3.2 Desvantagens

Apesar das vantagens, a estrutura também apresenta algumas desvantagens:

- **Inserção e Remoção:** A inserção ou remoção de um elemento no meio do array pode ser menos eficiente, pois exige o reposicionamento dos demais elementos.
- **Consumo de Memória:** Armazenar objetos inteiros pode consumir mais memória do que arrays de tipos de dados simples, embora para a escala deste projeto, isso não seja um problema significativo.

4 ANÁLISE DO ALGORITMO DE ORDENAÇÃO: SELECTION-SORT

O algoritmo Selection-Sort foi implementado para ordenar os dados dos alunos conforme os requisitos do sistema.

4.1 Por que o Selection-Sort?

O Selection-Sort foi escolhido por sua simplicidade e facilidade de compreensão. O algoritmo é intuitivo, pois funciona encontrando o menor (ou maior) elemento da lista e trocando-o para a posição correta, repetindo esse processo até que toda a lista esteja ordenada. Em um contexto didático, a sua lógica passo a passo torna o código claro para fins de demonstração, o que é um ponto forte para esta documentação.

4.2 Comparação com Outros Métodos

- **Bubble-Sort:** O Bubble-Sort, embora também seja simples de implementar, tende a ser menos eficiente que o Selection-Sort, pois realiza um número maior de trocas de elementos. O Selection-Sort minimiza o número de trocas, realizando apenas uma por iteração principal, o que pode ser uma pequena vantagem de desempenho.
- **Merge-Sort:** O Merge-Sort é um algoritmo de ordenação muito mais eficiente, especialmente para grandes volumes de dados. No entanto, sua implementação é recursiva e consideravelmente mais complexa. Para os objetivos deste projeto, que prioriza a clareza e o aprendizado, a complexidade do Merge-Sort seria desnecessária.
- **Busca (Sequencial e Binária):** Estes algoritmos não são de ordenação, mas sim de busca. A busca sequencial foi utilizada implicitamente na função de filtragem (.filter()) para encontrar os alunos aprovados, enquanto a busca binária não se aplica neste contexto, pois exige que os dados já estejam ordenados para funcionar.

5 APLICAÇÃO E TELAS

A aplicação foi desenvolvida com a biblioteca **React** em JavaScript, proporcionando uma interface de usuário dinâmica. A seguir, o código-fonte principal e os prints que demonstram o funcionamento do sistema.

5.1 Código-Fonte

A lógica do sistema está concentrada no arquivo App.jsx, que gerencia o estado da aplicação e as suas funcionalidades.

Figura 1 - Estrutura das pastas do projeto

A imagem acima ilustra a organização dos diretórios e arquivos do projeto, conforme a estrutura padrão de aplicações React inicializadas com Vite. Destaca-se a pasta src, que contém o App.jsx, index.css e main.jsx, onde reside o código-fonte principal da aplicação. A clareza na organização facilita a manutenção e a localização dos componentes.

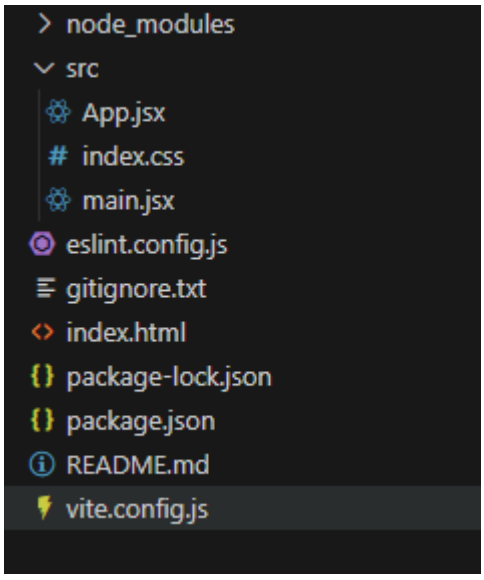


Figura 2 – Gerenciamento de estado e efeito.

Esta captura de tela do código App.jsx demonstra a declaração dos estados (useState) que gerenciam as variáveis da aplicação. São criados estados para os campos do formulário (nome, ra, idade, etc.), para o array principal de alunos (alunos), e para a lista que será exibida e ordenada (alunosExibidos). O uso de useState permite que o React monitore e atualize a interface automaticamente.

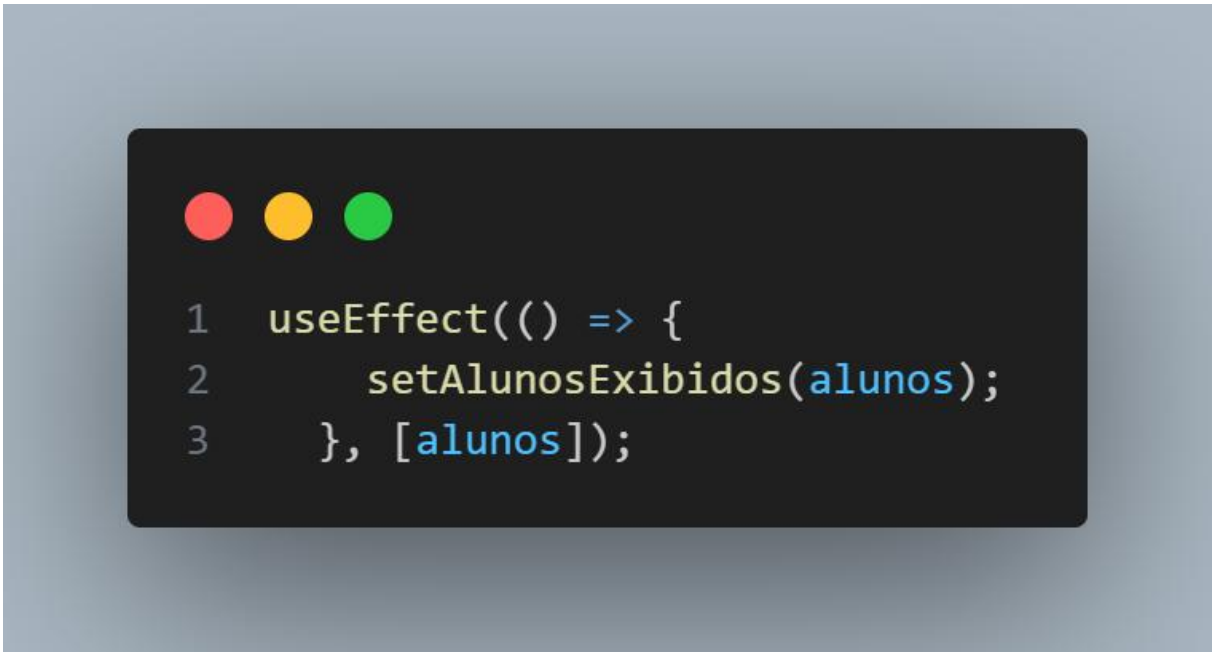
```
import { useState, useEffect } from "react";

function App() {
  const [nome, setNome] = useState("");
  const [ra, setRa] = useState("");
  const [idade, setIdade] = useState("");
  const [sexo, setSexo] = useState("");
  const [media, setMedia] = useState("");
  const [alunos, setAlunos] = useState([]);
  const [alunosExibidos, setAlunosExibidos] = useState([]);
  const [erros, setErros] = useState({});
  const [mensagemSucesso, setMensagemSucesso] = useState("");
  const [ordenacaoAtiva, setOrdenacaoAtiva] = useState(null);

```

Figura 3 - Sincronização de listas com useEffect.

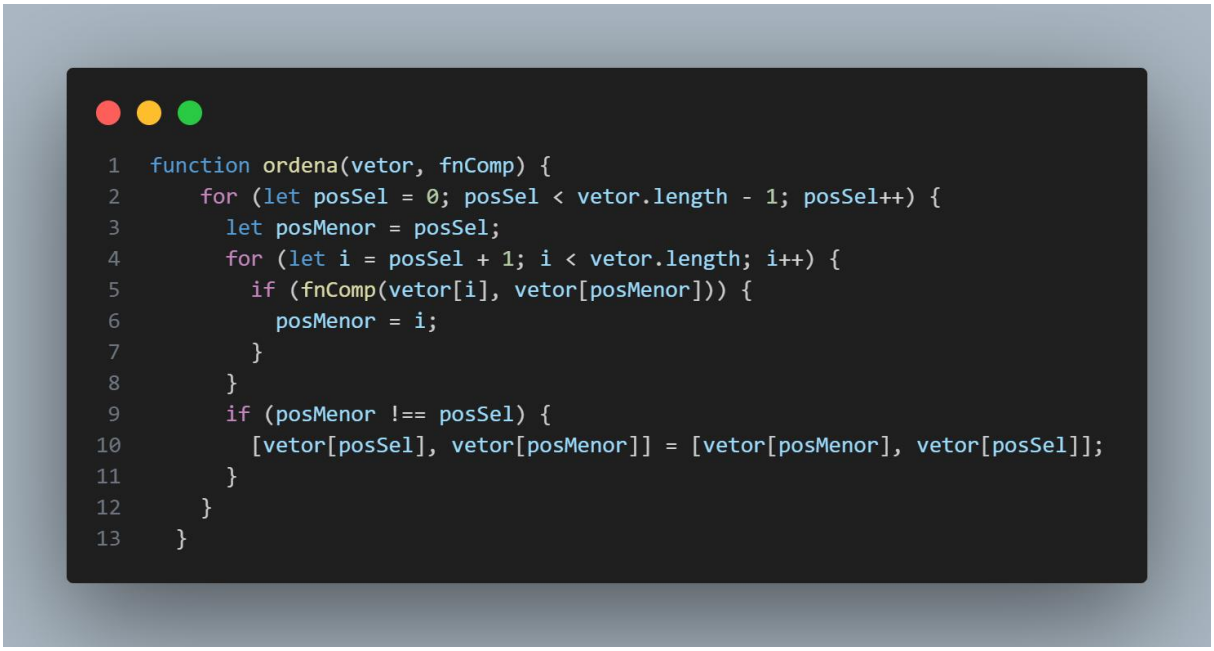
A função useEffect é empregada para garantir que a lista de alunos exibidos (alunosExibidos) seja automaticamente atualizada sempre que houver uma alteração na lista principal de alunos (alunos). Essa abordagem garante que os relatórios sempre exibam a lista mais recente de alunos cadastrados, mesmo após novas adições.



```
useEffect(() => {
    setAlunosExibidos(alunos);
  }, [alunos]);
```

Figura 4 – Implementação do Selection-Sort

Esta imagem detalha a implementação da função ordena, que materializa o algoritmo Selection-Sort. É uma função genérica, projetada para receber um vetor (o array a ser ordenado) e uma fnComp (uma função de comparação). A lógica do for externo percorre a lista e, no for interno, encontra o elemento correto para a posição, realizando a troca apenas no final de cada iteração.



```
function ordena(vetor, fnComp) {
    for (let posSel = 0; posSel < vetor.length - 1; posSel++) {
      let posMenor = posSel;
      for (let i = posSel + 1; i < vetor.length; i++) {
        if (fnComp(vetor[i], vetor[posMenor])) {
          posMenor = i;
        }
      }
      if (posMenor !== posSel) {
        [vetor[posSel], vetor[posMenor]] = [vetor[posMenor], vetor[posSel]];
      }
    }
  }
```

Figura 5 - Função ordenaNomeCrescente():

Esta função é responsável por ordenar a lista de alunos em ordem alfabética crescente por nome. Ela cria uma cópia do array original para não alterar os dados principais. Em seguida, chama a função ordena e passa uma função de comparação que avalia se o nome do primeiro elemento (elem1.nome) é menor que o do segundo (elem2.nome). O resultado é uma lista de alunos organizada de A a Z.

```
function ordenaNomeCrescente() {
    const ordenado = [...alunos];
    ordena(ordenado,(elem1, elem2) => elem1.nome.toLowerCase() < elem2.nome.toLowerCase());
    setAlunosExibidos(ordenado);
    setOrdenacaoAtiva("nome");
  }
```

Figura 6 – Função ordenaRaDecrescente():

Esta função ordena a lista de alunos em ordem numérica decrescente pelo RA. Similar à anterior, ela faz uma cópia do array e chama a função ordena, mas com uma função de comparação diferente. A lógica agora verifica se o RA do primeiro elemento (parseInt(elem1.ra)) é maior que o do segundo (parseInt(elem2.ra)), resultando em uma lista ordenada do maior RA para o menor.

```
function ordenaRaDecrescente() {
    const ordenado = [...alunos];
    ordena(ordenado, (elem1, elem2) => parseInt(elem1.ra) > parseInt(elem2.ra));
    setAlunosExibidos(ordenado);
    setOrdenacaoAtiva("ra");
  }
```

Figura 7 –Função ordenaNomeCrescenteAprovados():

Esta função demonstra a capacidade de combinar filtragem e ordenação. Primeiro, ela utiliza a função filter do JavaScript para criar um novo array contendo apenas os alunos aprovados (aqueles com média igual ou superior a 6,0). Depois, ela chama a função ordena para organizar essa lista filtrada em ordem crescente por nome, fornecendo um relatório preciso e específico.

```
function ordenaNomeCrescenteAprovados() {
  const aprovados = alunos.filter((a) => a.resultado === "Aprovado");
  const ordenado = [...aprovados];
  ordena(ordenado,(elem1, elem2) => elem1.nome.toLowerCase() < elem2.nome.toLowerCase());
  setAlunosExibidos(ordenado);
  setOrdenacaoAtiva("aprovados");
}
```

Figura 8 – Lógica de submissão e validação

A função handleSubmit é ativada no envio do formulário. Ela é responsável por realizar a validação dos campos, verificando se estão preenchidos corretamente. Em caso de erros, o processo é interrompido. Esta etapa é crucial para garantir a integridade dos dados antes de adicioná-los à estrutura de dados.

```
const handleSubmit = (e) => {
    e.preventDefault();

    // Validação de campos obrigatórios
    const novosErros = {};
    if (!nome) novosErros.nome = "Campo obrigatório";
    if (!ra) novosErros.ra = "Campo obrigatório";
    if (!idade) novosErros.idade = "Campo obrigatório";
    if (!sexo) novosErros.sexo = "Campo obrigatório";
    if (!media) {
      novosErros.media = "Campo obrigatório";
    } else {
      const mediaNumerica = parseFloat(media);
      // Validação de média
      if (
        mediaNumerica < 0 ||
        mediaNumerica > 10 ||
        mediaNumerica !== mediaNumerica
      ) {
        novosErros.media = "A média deve ser um número entre 0 e 10.";
      }
    }

    setErros(novosErros);
    // Se houver algum erro, interrompe o envio do formulário
    if (Object.keys(novosErros).length > 0) {
      setMensagemSucesso(""); // Limpa a mensagem de sucesso se houver erro
      return;
    }

```

Figura 9 - Criação e adição de um novo aluno

A parte final da função handleSubmit ilustra a criação de um novo **objeto de aluno**. Os dados do formulário são usados para preencher as propriedades (nome, ra, idade, etc.). O campo resultado é calculado automaticamente com base na média. Finalmente, o novo objeto é adicionado ao array de alunos (setAlunos([...alunos, novoAluno])), expandindo a estrutura de dados de forma dinâmica.

```
const nomeNormalizado =
      nome.charAt(0).toUpperCase() + nome.slice(1).toLowerCase();
    const novoAluno = {
      nome: nomeNormalizado,
      ra: ra,
      idade: parseInt(idade),
      sexo,
      media: parseFloat(media),
      resultado: parseFloat(media) >= 6.0 ? "Aprovado" : "Reprovado",
    };

    setAlunos([...alunos, novoAluno]);
    setMensagemSucesso("Aluno cadastrado com sucesso!");
    setTimeout(() => {
      setMensagemSucesso("");
    }, 3000);
    setNome("");
    setRa("");
    setIdade("");
    setSexo("");
    setMedia("");
  };
```

Figura 10 - Seção de Cadastro

Esta parte do código cria o formulário HTML (<form>) com todos os campos de entrada, as etiquetas (<label>) e o botão de submissão. Ele também gerencia a exibição de mensagens de sucesso e erros de validação, tornando a interação mais amigável.

```jsx
<div className="w-full max-w-md my-4">
          <div className="bg-white shadow-lg rounded-2xl p-8">
            <p className="text-xl font-bold text-gray-700 mb-4 text-center">
              Relatórios de Alunos
            </p>
            <div className="mt-4 mx-3 flex justify-between gap-3">
              <button
                onClick={ordenaNomeCrescente}
                className={`flex items-center bg-blue-200 text-black py-2 px-4 rounded-lg hover:bg-blue-300 transition cursor-pointer
                  ${ordenacaoAtiva === "nome" ? "bg-blue-400 text-white" : ""}`}
              >
                <span className="mr-1">↑</span> Nome
              </button>
              <button
                onClick={ordenaRaDecrescente}
                className={`flex items-center bg-blue-200 text-black py-2 px-4 rounded-lg hover:bg-blue-300 transition cursor-pointer
                  ${ordenacaoAtiva === "ra" ? "bg-blue-400 text-white" : ""}`}
              >
                <span className="mr-1">↓</span> RA
              </button>
              <button
                onClick={ordenaNomeCrescenteAprovados}
                className={`flex items-center bg-blue-200 text-black py-2 px-4 rounded-lg hover:bg-blue-300 transition cursor-pointer
                  ${ordenacaoAtiva === "aprovados"? "bg-blue-400 text-white": ""}`}
              >
                <span className="mr-1">↑</span> Aprovados
              </button>
            </div>
            <div className="mt-8 w-full max-w-md space-y-4">
              {alunosExibidos.length === 0 ? (
                <div className="text-center text-gray-500 text-sm">
                  Nenhum aluno cadastrado. Adicione o primeiro para ver a lista.
                </div>
              ) : (alunosExibidos.map((aluno, index) => (
                  <div
                    key={index}
                    className="bg-white shadow-md rounded-lg p-4 border border-gray-200 transition-shadow duration-300 hover:shadow-lg"
                  >
                    <h2 className="text-lg font-semibold text-gray-700">
                      {" "}
                      {aluno.nome}
                    </h2>
                    <div className="mt-2 text-sm text-gray-600 space-y-1">
                      <p>
                        <span className="font-medium">RA:</span> {aluno.ra}
                      </p>
                      <p>
                        <span className="font-medium">Idade:</span>{" "}
                        {aluno.idade}
                      </p>
                      <p>
                        <span className="font-medium">Sexo:</span> {aluno.sexo}
                      </p>
                      <p>
                        <span className="font-medium">Média:</span>{" "}
                        {aluno.media}
                      </p>
                      <p>
                        <span className="font-medium">Resultado: </span>
                        <span className="font-medium">{aluno.resultado}</span>
                      </p>
                    </div>
                  </div>
                ))
              )}
            </div>
          </div>
        </div>
      </div>
    </div>
```

Figura 11 - Seção de Relatórios

Esta parte do código renderiza os botões de ordenação e a lista de alunos. É aqui que a mágica acontece na interface: a lista é dinamicamente exibida com base no array alunosExibidos, que é manipulado pelas funções de ordenação.

```jsx
<div className="w-full max-w-md my-4">
          <div className="bg-white shadow-lg rounded-2xl p-8">
            <p className="text-xl font-bold text-gray-700 mb-4 text-center">
              Cadastro de Alunos
            </p>
            <form onSubmit={handleSubmit} className="space-y-4">
              {mensagemSucesso && (
                <div className="bg-green-100 border border-green-400 text-green-700 px-4 py-3 rounded relative mb-4">
                  <span className="block sm:inline">{mensagemSucesso}</span>
                </div>
              )}
              <div>
                <label className="block text-gray-600">Nome: </label>
                <input
                  placeholder="Digite seu nome..."
                  value={nome}
                  onChange={(e) => {setNome(e.target.value)
                  if (erros.nome) {
                      setErros({ ...erros, nome: null });
                    }
                  }}
                  className={`w-full px-3 py-2 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-400 ${
                    erros.nome ? "border-red-500 border-2" : "border-gray-300"
                  }`}
                />
                {erros.nome && (<p className="text-red-500 text-sm mt-1">{erros.nome}</p>)}
              </div>
              <div>
                <label className="block text-gray-600">RA: </label>
                <input
                  placeholder="Digite seu RA..."
                  value={ra}
                  onChange={(e) => {setRa(e.target.value)
                    if (erros.ra) {
                        setErros({ ...erros, ra: null });
                    }
                  }}
                  className={`w-full px-3 py-2 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-400
                    ${erros.ra ? "border-red-500 border-2" : "border-gray-300"}`}
                />
                {erros.ra && (<p className="text-red-500 text-sm mt-1">{erros.ra}</p>)}
              </div>
              <div>
                <label className="block text-gray-600">Idade: </label>
                <input
                  placeholder="Digite sua idade..."
                  value={idade}
                  onChange={(e) => {setIdade(e.target.value);
                    if (erros.idade) {
                      setErros({ ...erros, idade: null });
                    }
                  }}
                  className={`w-full px-3 py-2 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-400
                    ${erros.idade ? "border-red-500 border-2" : "border-gray-300"}`}
                />
                {erros.idade && (<p className="text-red-500 text-sm mt-1">{erros.idade}</p>)}
              </div>
              <div>
                <label className="block text-gray-600">Sexo: </label>
                <select
                  value={sexo}
                  onChange={(e) => {
                    setSexo(e.target.value);
                    if (erros.sexo) {
                      setErros({ ...erros, sexo: null });
                    }
                  }}
                  className={`w-full px-3 py-2 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-400
                    ${erros.sexo ? "border-red-500 border-2" : "border-gray-300"}`}
                >
                  <option value="">Selecione...</option>
                  <option value="Feminino">Feminino</option>
                  <option value="Masculino">Masculino</option>
                </select>
                {erros.sexo && (<p className="text-red-500 text-sm mt-1">{erros.sexo}</p>)}
              </div>
              <div>
                <label className="block text-gray-600">Média: </label>
                <input
                  placeholder="Digite a média..."
                  value={media}
                  onChange={(e) => {
                    setMedia(e.target.value);
                    if (erros.media) {
                      setErros({ ...erros, media: null });
                    }
                  }}
                  className={`w-full px-3 py-2 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-400
                    ${erros.media ? "border-red-500 border-2" : "border-gray-300"}`}
                />
                {erros.media && (<p className="text-red-500 text-sm mt-1">{erros.media}</p>)}
              </div>
              <button
                type="submit"
                className="w-full bg-blue-500 text-white py-2 rounded-lg hover:bg-blue-600 transition cursor-pointer"
              >
                Adicionar Aluno
              </button>
            </form>
          </div>
        </div>
```

5.2 Prints das Telas

As imagens a seguir demonstram o funcionamento da aplicação, desde a sua interface inicial até as interações de validação, cadastro e geração de relatórios.

Figura 12 – Tela inicial da aplicação

Apresenta a interface da aplicação ao ser acessada pela primeira vez, mostrando a estrutura de duas colunas para o cadastro de alunos e a área de relatórios, que inicialmente exibe uma mensagem de lista vazia.

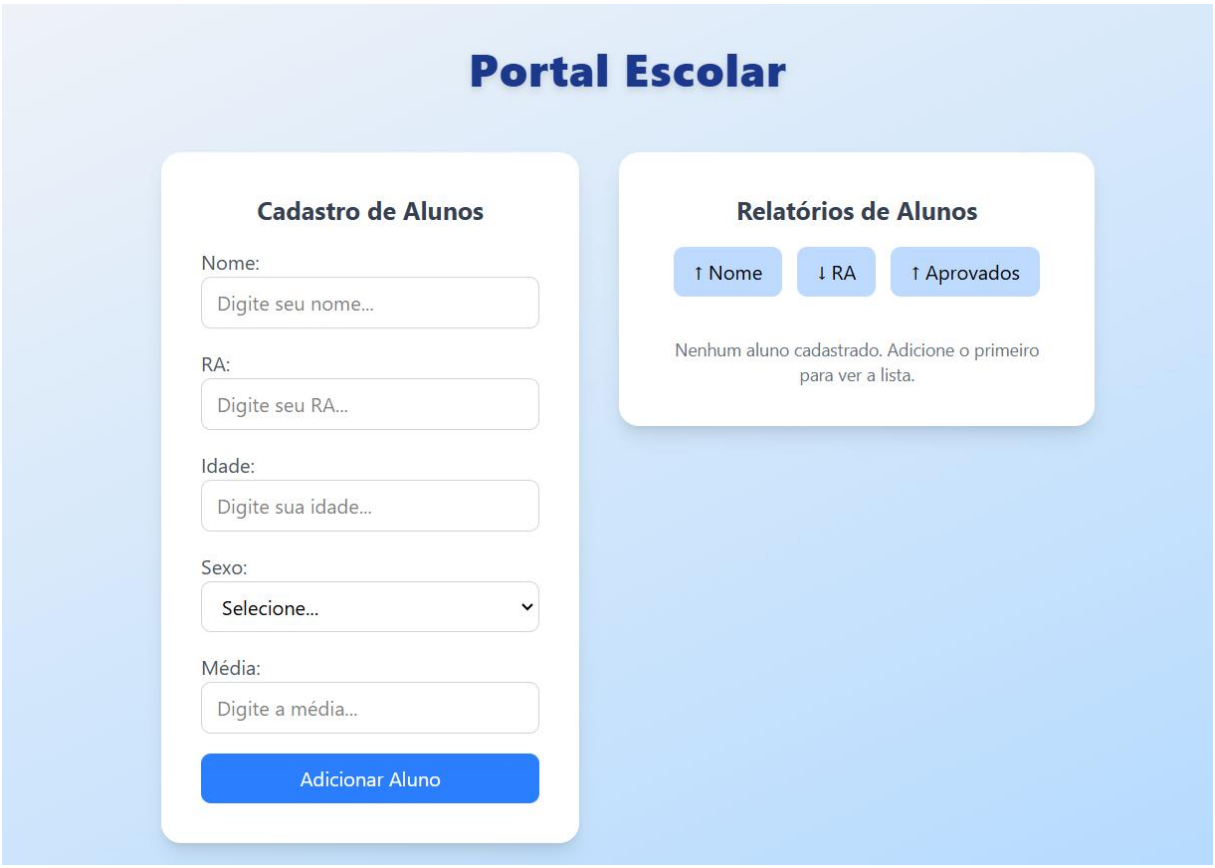


Figura 13 – Validação de dados e tratamento de erros

Esta imagem ilustra a etapa de validação do formulário. Quando o usuário tenta submeter o formulário sem preencher os campos obrigatórios, o sistema exibe mensagens de erro em tempo real, garantindo a integridade dos dados antes do cadastro.

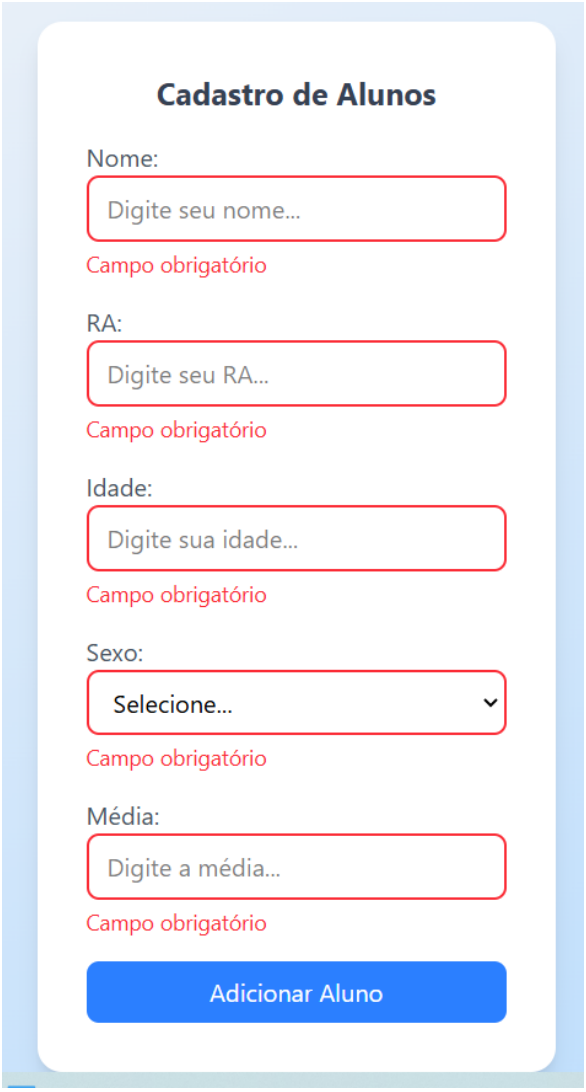


Figura 14 – Cadastro de aluno com sucesso

A imagem demonstra o processo completo de cadastro. Após o preenchimento correto dos dados e a submissão do formulário, a aplicação exibe uma mensagem de sucesso e adiciona o novo aluno à lista, expandindo a estrutura de dados dinamicamente.

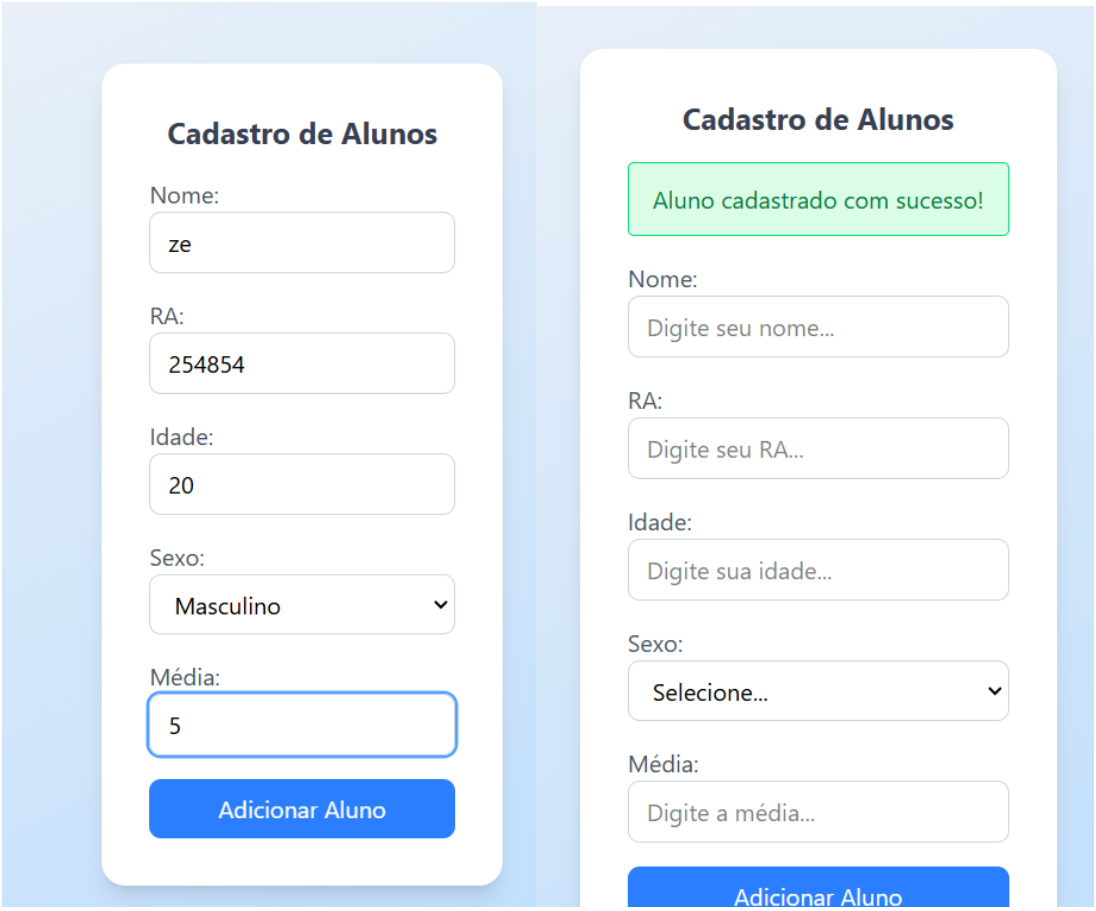
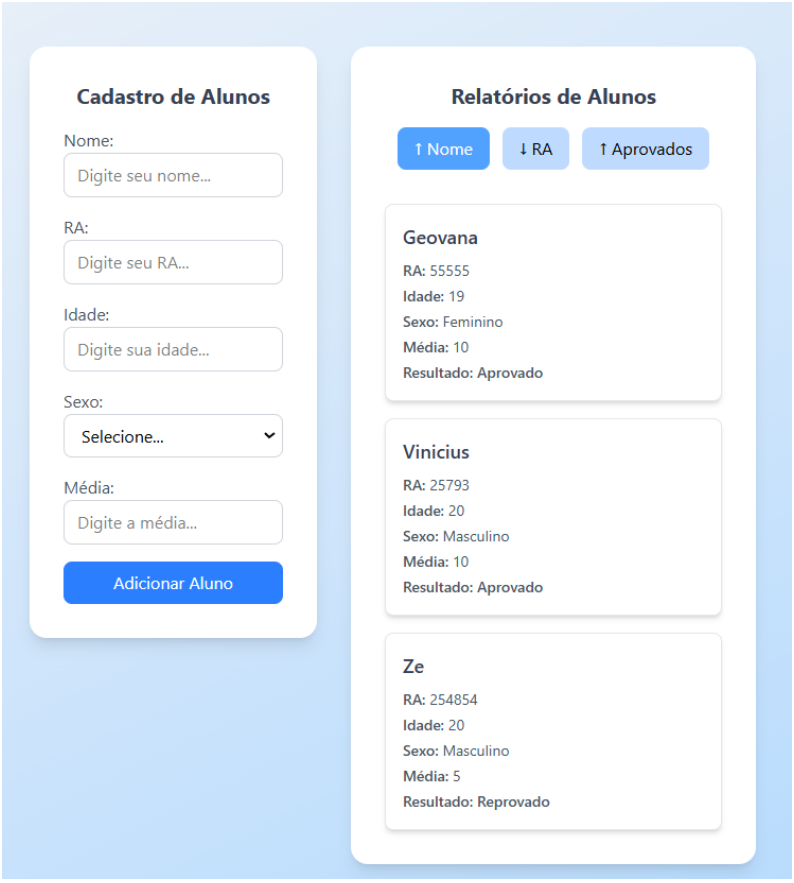
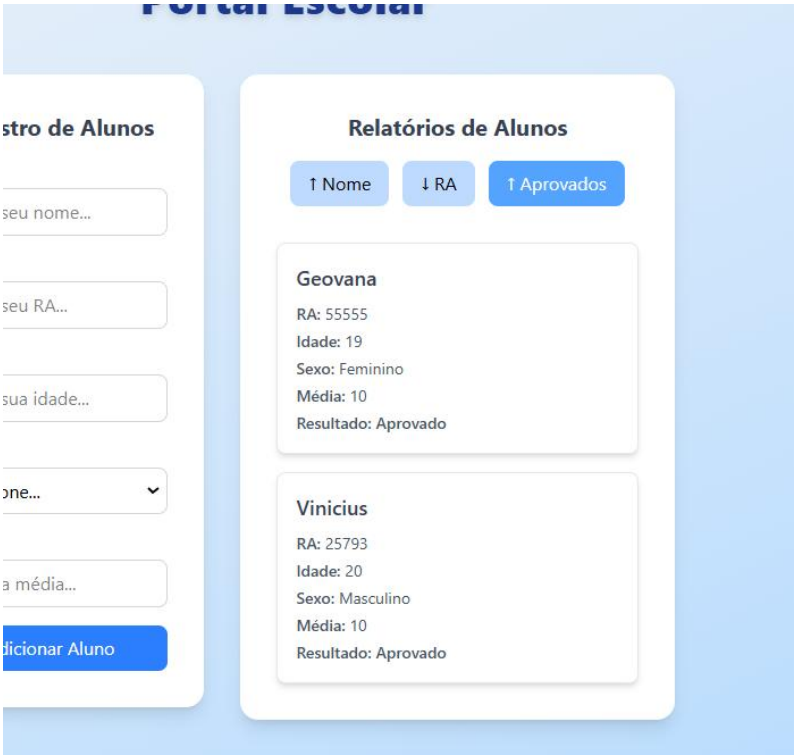
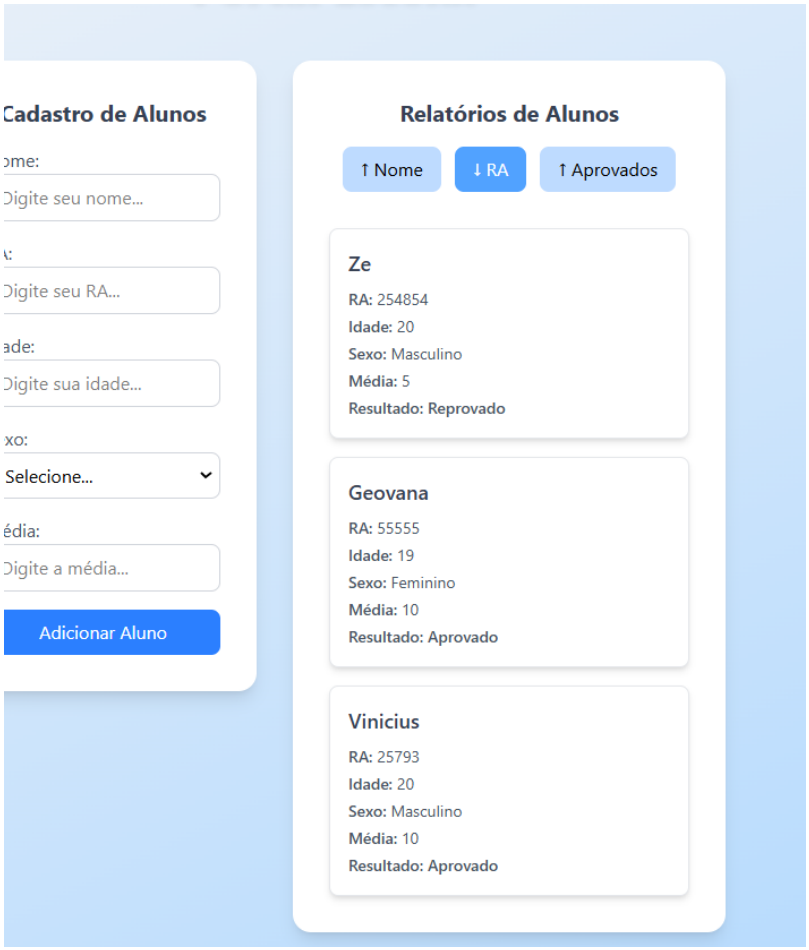


Figura 15 – Relatórios e ordenação.

Esta tela exibe a funcionalidade de relatórios da aplicação. A imagem mostra a lista de alunos ordenada de acordo com um dos critérios disponíveis. O sistema também é capaz de gerar relatórios em ordem crescente por nome e relatórios filtrados para exibir apenas os alunos aprovados, todos utilizando o algoritmo Selection-Sort.


Cadastro de Alunos
Nome:
Digite seu nome...
RA:
Digite seu RA...
Idade:
Digite sua idade...
Sexo:
Selecione...
Média:
Digite a média...
Adicionar Aluno
Relatórios de Alunos
↑ Nome
↓ RA
↑ Aprovados
Geovana
RA: 55555
Idade: 19
Sexo: Feminino
Média: 10
Resultado: Aprovado
Vinicius
RA: 25793
Idade: 20
Sexo: Masculino
Média: 10
Resultado: Aprovado
Ze
RA: 254854
Idade: 20
Sexo: Masculino
Média: 5
Resultado: Reprovado


Relatórios de Alunos
↑ Nome
↓ RA
↑ Aprovados
Geovana
RA: 55555
Idade: 19
Sexo: Feminino
Média: 10
Resultado: Aprovado
Vinicius
RA: 25793
Idade: 20
Sexo: Masculino
Média: 10
Resultado: Aprovado


Cadastro de Alunos
ome:
Digite seu nome...
:
Digite seu RA...
ade:
Digite sua idade...
xo:
Selecione...
édia:
Digite a média...
Adicionar Aluno
Relatórios de Alunos
↑ Nome
↓ RA
↑ Aprovados
Ze
RA: 254854
Idade: 20
Sexo: Masculino
Média: 5
Resultado: Reprovado
Geovana
RA: 55555
Idade: 19
Sexo: Feminino
Média: 10
Resultado: Aprovado
Vinicius
RA: 25793
Idade: 20
Sexo: Masculino
Média: 10
Resultado: Aprovado

6 CONSIDERAÇÕES FINAIS

O desenvolvimento desta aplicação permitiu a aplicação prática de conceitos teóricos de estruturas de dados e algoritmos. O uso do Array de Objetos mostrou-se eficaz para o armazenamento de dados heterogêneos de forma flexível e organizada. A implementação do algoritmo Selection-Sort, apesar de não ser a mais rápida para grandes volumes de dados, cumpriu seu papel de forma eficiente para os requisitos do projeto, priorizando a simplicidade e a clareza do código. O software final demonstrou a capacidade de realizar cadastro, ordenação e filtragem de dados, comprovando a robustez da solução proposta e o domínio dos conceitos estudados.

7 CONSIDERAÇÕES FINAIS

Retomar o objetivo geral e resumir os procedimentos realizados, as possibilidades de outros estudos e as possíveis fragilidades da atividade.

8 CONTRIBUIÇÕES DA UCE PARA A FORMAÇÃO DISCENTE

Relatar a relevância do desenvolvimento da atividade para a própria formação, considerando o objetivo da UCE. Pode-se considerar o parecer do grupo que executou as atividades.